AN ARCHITECTURE FOR INTEGRATING DISTRIBUTED AND COOPERATING KNOWLEDGE-BASED AIR FORCE DECISION AIDS

Richard O. Nugent
Nugent@MITRE.Arpa

Richard W. Tucker
RWTucker@MITRE.Arpa

The **MITRE** Corporation
Washington C^3I Artificial Intelligence Technical Center
7525 Colshire Drive
McLean, Virginia 22102

ABSTRACT

MITRE has been developing a Knowledge-Based Battle Management Testbed for evaluating the viability of integrating independently-developed knowledge-based decision aids in the Air Force tactical domain.

The primary goal for the testbed architecture is to permit a new system to be added to a testbed with little change to the system's software. Each system that connects to the testbed network declares that it can provide a number of services to other systems. When a system wants to use another system's service, it does not address the server system by name, but instead transmits a request to the testbed network asking for a particular service to be performed.

A key component of the testbed architecture is a common database which uses a relational database management system. The RDBMS provides a database update notification service to requesting systems. Normally, each system is expected to monitor data relations of interest to it. Alternatively, a system may broadcast an announcement message to inform other systems that an event of potential interest has occurred.

Current research is aimed at dealing with issues resulting from integration efforts, such as dealing with potential mismatches of each system's assumptions about the common database, decentralizing network control, and coordinating multiple agents.

INTRODUCTION

Integrating heterogeneous software systems is a burgeoning problem, particularly for the military. Many independently-developed systems produced for the military are stand-alone decision aids. This paper describes an architecture which supports the integration of such command and control (C^2) systems and discusses the required characteristics which enable these systems to cooperate and share information with each other.

MITRE's Knowledge-Based Battle Management Testbed has been the vehicle for performing experiments in integrating knowledge-based systems for the Rome Air Development Center (RADC) [5]. The testbed employs a core set of functions which provide control mechanisms and open connectivity support, called the knowledge-based battle management (KB-BATMAN) shell. The type of systems for which the testbed is intended are coarse-grained, loosely-coupled systems. A coarse-grained system has a large amount of functionality, and a loosely-coupled system has a high level of independence from other systems; such a system does not require a great deal of communication with external agents, and can act autonomously most of the time. A primary goal of the testbed architecture has been to permit systems to be able to "plug in" dynamically and even be replaceable by systems offering similar functionality.

Three realistic Air Force tactical C^2 systems operate in the current testbed: a mission planner, a simulator, and an intelligence analysis system. The goal of the testbed project has been to link these three coarse-grained systems using the KB-BATMAN Shell and to determine what problems must be addressed to assure effective cooperation among them. The principal way in which these three systems are linked is by relaying outputs from one system, such as the intelligence analyst, to become inputs to another system, such as the mission planner. This concept of integration seems simple; however, a variety of issues are involved, some of which have been addressed during the testbed project.

The principal issues that have been addressed include how to control cooperation, how to permit commonly-used information to be used by several systems, and how to deal with different views or representations of information.

BACKGROUND AND PROBLEMS

Related Research

It is difficult to evaluate the effectiveness of a decision aid in isolation from other systems with which it may interact. Graham describes a model for representing the interaction of systems with their environment, where the environment is the essential "glue" through which the systems interact [4].

Graham views the environment as one more system to be modeled in a distributed simulation of C^2 systems.

Previous MITRE work on the AirLand Loosely Integrated Expert Systems (ALLIES) project [1] involved integrating an Army planning system, an intelligence analysis system, and a simulation system into a single cooperating environment. Since these systems were integrated after each was developed to operate stand-alone, the methodology for integration was ad hoc and communications required several different protocols.

A better environment for developing cooperating, distributed systems is essential to encourage modularity of system design and to provide well-defined interfaces among systems. Teknowledge, Inc. has been developing ABE for RADC and the Defense Advanced Research Projects Agency (DARPA) to meet these goals [3]. ABE was not used in the testbed project because it was still under development when evaluated.

Integration of heterogeneous decision aids requires addressing issues involving the fields of distributed computing, databases, networking, and knowledge-based systems, among others.

Distributed Control

In any distributed environment, control of intersystem activities may be centralized or decentralized, or a hybrid. Centralized control is easiest to implement, but also provides a single point of failure, which would not be desirable in an operational system in most cases. Decentralized control requires fairly complex algorithms for coordination of systems. We use centralized control in our testbed, in part to keep the architecture simple, and also to support monitoring of testbed communications.

Common Functionality

While heterogeneous software components should be loosely-coupled to prevent each system from becoming highly dependent on other systems, there still is a need for sharing information that is not specific to a single system. Two common systems have been identified to fulfill this requirement: a Common Database manager and a Common Knowledge Base manager. These two components are considered to be integral parts of the KB-BATMAN Shell, although like other systems in the testbed, they are modules which can be replaced without affecting other systems.

A relational database management system (RDBMS) is used for the Common Database because the functionality of RDBMSs are fairly standard and implementations are available for a wide variety of computers. Most properly-designed decision aids should have an easily identifiable set of database access functions which may be replaced with RDBMS functions accessing a Common Database.

When a new system is being integrated into the testbed, it is important to determine how it uses a database. One problem is to decide which data elements are of interest to other systems in the testbed, and which data elements are for internal use only. A second-order problem is to ascertain how to translate data representations into a form that is most appropriate for access by multiple systems, since different systems may view the same collection of data in different ways. Each system's view of the organization of data must be transformed to the Common Database's actual view. To solve this problem, the Common Database system must be able to provide an intelligent database viewing mechanism, in which a database request from a system may be translated to a combination of select, join, and project operations in order to provide the requested view. The alternative to providing intelligent interfaces is to modify the internal structure of a system, which is likely to be an undesirable option for large-scale, coarse-grained systems.

There are further issues resulting from multiple access to commonly-used information. One system may use a different method of representing some entity; for example in the testbed, one system uses latitude and longitude to identify a ground location whereas another system uses the universal transverse Mercator coordinate system. Also, one system may be interested in greater precision or detail for some data than needed by another system. Interpretation of uncertainty qualifications to data is likely to be difficult or impossible to correlate between systems.

The Common Knowledge Base includes commonly-needed behaviors or functionality for the Air Force problem domain. It can be used to reduce the duplication of effort in component systems. It can also be used to enforce standard operating procedures as well as Air Force doctrine. Further work remains to be done on the Common Knowledge Base, particularly for its potential role as an overall director for a suite of C^2 systems.

Impact on Using Existing Systems

An early goal of the testbed project was to address the issue of using existing decision aid systems. It is impractical to suggest that any existing system can be easily adapted for integration into our testbed architecture. Systems which were not designed with integration in mind are especially likely to be difficult to adapt. It may be more cost-effective and reliable to reimplement a system to fit the architecture than to patch existing software.

THE KB-BATMAN SHELL ARCHITECTURE

Message Passing

Testbed components communicate with each other by sending messages. Three types of messages are used: a request, a reply, and a notification. A request corresponds to a the concept of remote *procedure call* from distributed computing. A reply contains data in response to a request. A notification is an announcement which does not imply that a reply is expected.

ORIGINAL PAGE IS
OF POOR QUALITY

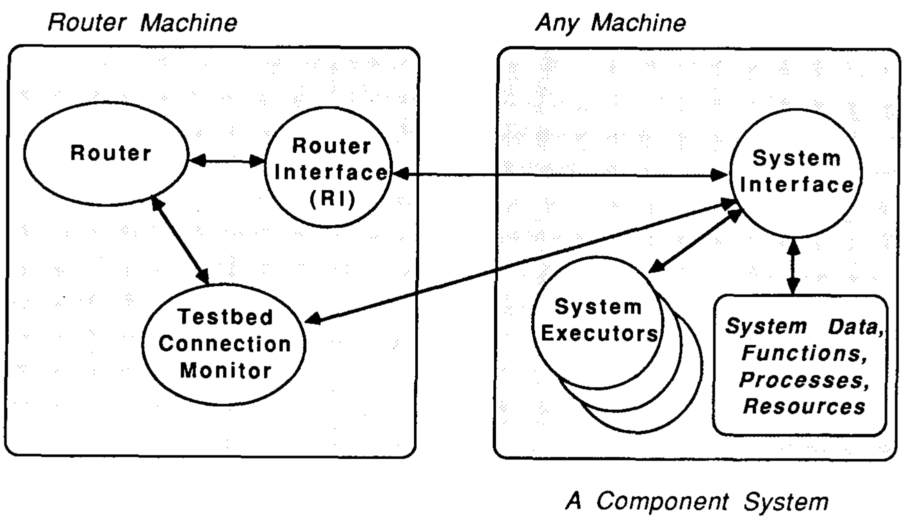

Figure 1

Processes

Multiple processes are involved in the KB-BATMAN shell concept. The following types of processes are used in the testbed:

- The Router
  (one process)
- Testbed Connection Monitor
  (one process)
- Router Interfaces (RIs)
  (one per hosted system)
- System Interfaces (SIs)
  (one per hosted system)
- System Executors
  (one per active request execution)

Figure 1 shows the interrelationships among these processes

The Router supports centralized control of communications among hosted systems. All messages pass through the Router. Figure 2 depicts the star network of systems communicating through the Router.

A Router Interface and System Interface together form the communications interface between a system and the Router. The interface consists of two parts because the system may execute on a different computer than the Router. A system's RI process executes on the same machine as the Router; the SI process can operate on any computer, but typically is associated with a machine representing the hosted system. (A hosted system itself may operate on multiple computers). Processes on both computers need to poll for message arrival from either side, either from the system or the Router.

The Testbed Connection Monitor operates on the same computer as the Router. Its purpose is to handle requests from systems on other machines to connect themselves into the testbed. The Connection Monitor asks the Router process to create an RI for the system. The RI and SI then will open the necessary network connections to support message passing.

Through its SI, each system declares to the Router the services it can perform upon request. For example, the Common Database system advertises that it will service database access and database update notification requests. In addition, "declare services" is a built-in service handled by the Router.

When a system's SI receives a request for one of its services from another system via the Router, it executes that request by evaluating a function asynchronously in a separate System Executor process. In other words, the service request can be in execution while the SI continues to poll for further messages; in fact, multiple service requests can be in execution, each in a separate System Executor process.

The Router's role is to maintain a "yellow pages" of all declared services. It is possible that a service can be performed by more than one system. When a system requests an external service, its SI sends the request to its RI which relays it to the Router. The Router determines which system is most appropriate to perform the service by selecting one from the set of all declared servers. (In the present implementation, no criteria are applied for selecting from multiple servers. Criteria might include speed of response, accuracy of response, currency of data, etc.) The service requester does not address its request to a particular system; in fact, the requester does not know what system, if any, will perform a service. It is possible that a service is not supported, in which case the Router sends an error indication as a reply to the requester. The Router currently does not interpret the contents of messages.

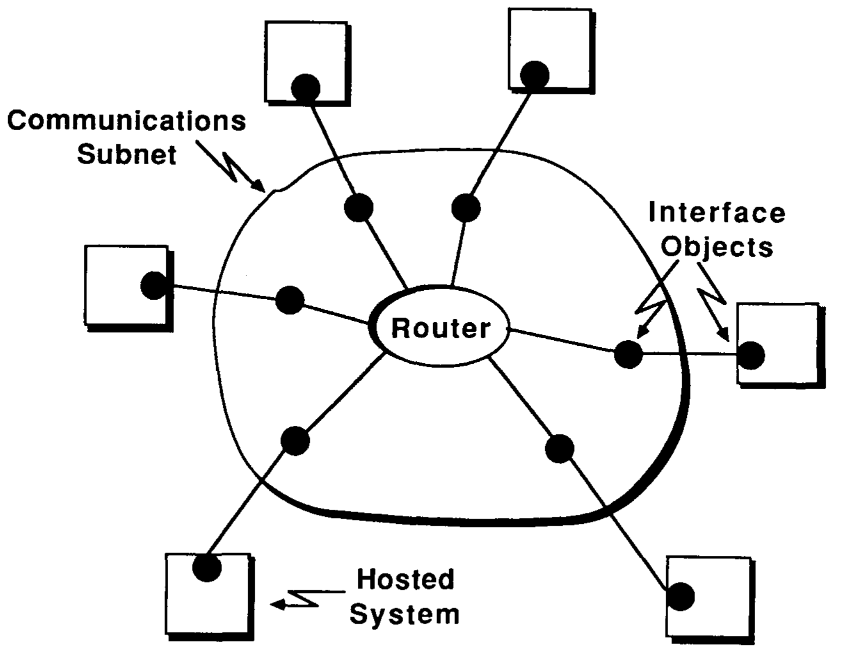

Figure 2

A System Executor process is created by an SI to perform a requested service. The System Executor can be considered to be performing the service in the context of the SI's system since it has access to any functions and data of that system. System Executors are implemented as reusable process resources on a LISP machine, so after one completes the servicing of a request it becomes eligible for reuse.

Consider a situation in which a service handling system does not respond to a service request in a timely fashion, either due to system failure or because it is busy doing other things. The requesting system would wait a long time or even forever unless it is designed to monitor for replies and eventually time out. In general, it is preferable for distributed systems to be data-driven, responding to changes introduced by external sources (such as other systems or an operator), rather than function-driven, in which the system asks another system to perform a function and then waits for a response. A data-driven system is easier to coordinate with other systems than a function-driven one because it is reactive rather than dependent. Nevertheless, function-driven or service-oriented systems are necessary for a variety of general-purpose common functions that multiple systems may need to use, such as accessing the Common Database and using resource managers.

Input Ports

When a message is transmitted from one testbed process to another, it is stored in a process' input port. Each process other than System Executors has one input port, and the process may read messages from the input port in any order it chooses. Presently, most processes read and process all messages in a first-in, first-out (FIFO) manner. However, a System Executor process scans the contents of the input port of its currently associated SI for whichever replies it is expecting. A System Executor does not process any request messages, and the System Interface does not process any reply messages. Each process which monitors a network connection for input will copy network input messages into its input port for subsequent processing.

TAC-2 APPLICATION SYSTEMS

TAC-2 is the name of the version of the testbed which incorporates three realistic knowledge-based systems in the Air Force tactical domain: a mission planning system, an intelligence analysis system, and a simulation system. These three systems were developed by three different groups of people at different locations. One, the planning system, was under development prior to the initiation of TAC-2, whereas the other two were developed expressly for use with TAC-2.

The planning system used in TAC-2 is the Air Force Mission Planning system (AMPS), which is a successor to the KNOBS Replanning System (KRS) [2] which in turn was a successor to the Knowledge Based System (KNOBS). All of these systems have been developed by the MITRE-Bedford Artificial Intelligence Technical Center independently of the TAC-2 effort underway at the MITRE-Washington Artificial Intelligence Technical Center. The concepts embodied in KRS have also led to the current development of TEMPLAR for use as a operational system by the Air Force.

AMPS was developed as a stand-alone system, and none of the AMPS software was developed for use in TAC-2. However, one component of AMPS, its relational database management system, was adapted for use as part of TAC-2's Common Database system. A number of enhancements were made to the DBMS in order to support simple methods for remote access and update notification. The Common Database system is used to manage data for all component systems in TAC-2.

The intelligence system in TAC-2, INTEL, was developed by the same staff at MITRE-Washington that developed TAC-2 and therefore was easiest to adapt to the conventions required for inclusion in the TAC-2 testbed.

The simulation system in TAC-2, SIMULATOR, was developed by staff at the Rome Air Development Center. SIMULATOR was designed to work with TAC-2, but included some differences with the other systems in assumptions about data. For example, SIMULATOR assumes that location data is given in universal transverse Mercator coordinates whereas AMPS assumes that location data are given as latitude/longitude pairs. The transformation between the two location representations is complex. How can systems cooperate if they have differences like this? An intelligent interface to the Common Database must transparently supply the correct data to each system.

These three domain systems cooperate by reading and writing data to the Common Database. AMPS plans offensive counter-air (OCA) missions automatically based on target and other data present in the Common Database, built-in planning constraints, and optional user inputs. The Simulator simulates flying these missions, assessing bomb damage to the targets and loss of aircraft due to surface-to-air missiles (SAMs). The INTEL system prioritizes targets for bombing missions.

All TAC-2 component systems and support software are written in LISP (both Common LISP and ZETALISP) and run on Symbolics LISP machines. INTEL and SIMULATOR operate in the latest version of the operating system software, whereas AMPS executes in an earlier version. The KB-BATMAN shell software and the Common Database system software operate in both versions, using the same source code. Communications between LISP machines employs generic networking software, and can use either TCP/IP or Chaosnet protocols for transmission of messages between a system and the KB-BATMAN Router. TAC-2 can operate on as many as five computers, with each of the Router, Common Database, AMPS, INTEL, and SIMULATOR on a separate computer; or as few as two, with AMPS on one computer and the others all on another computer.

SUMMARY AND CONCLUSIONS

The KB-BATMAN Shell architecture provides support for integrating coarse-grained knowledge-based decision aids. The most important aspect of the architecture is the emphasis on maintaining independence of systems. Independent systems can be considered more manageable and robust than systems that rely on other systems for control directives. Independence is encouraged through the use of intersystem messages which are *not* addressed to specific systems. Instead, messages are either service-oriented, to be relayed by a Router to a system which supports the service, or broadcast into the environment for all systems to examine. A message may be nothing more than a piece of data in a shared Common Database, in which case system control is totally data-driven.

Another key aspect of the architecture is the use of intelligent interfaces to systems. Intelligent interfaces can be used to adapt data from the external environment (e.g., the Common Database) to be in a form suitable for internal system use. These interfaces may need to employ knowledge-based techniques for transforming data from a common representation to a specialized one used within the system. If necessary, an interface can interact with the environment and other systems in order to support its system's needs.

Further work is required to address issue areas such as decentralizing control away from a single Router and supporting component connectivities other than a star network. The Router needs to be extended to permit servers to provide qualifications for providing a service, and on the other end, to permit servers to be able to preview a request to determine whether it is interested in servicing it. The latter improvement would be necessary for a totally decentralized control mechanism such as contract nets, in which systems bid on service requests.

ACKNOWLEDGMENTS

This document was completed under Air Force Contract F19628-86-C-0001 in support of the Rome Air Development Center, Griffiss Air Force Base, New York.

REFERENCES

1. Benoit, John W. et al., *AirLand Loosely Integrated Expert Systems: The ALLIES Project*, MTR-86W00041, The MITRE Corporation, McLean VA, April 1986.

2. Dawson, Bruce C., Richard H. Brown, Candace E. Kalish, and Stuart Goldkind, *Knowledge-Based Replanning System*, RADC-TR-87-60, Rome Air Development Center, Griffiss Air Force Base NY, May 1987.

3. Erman, Lee D., Jay S. Lark, and Frederick Hayes-Roth, *Engineering Intelligent Systems: Progress Report on ABE*, TTR-ISE-86-102, Teknowledge, Inc., Palo Alto CA, May 1986.

4. Graham, Richard A., *An Environment for Distributed Simulation of Command and Control Networks*, Master of Science Thesis, Naval Postgraduate School, Monterey CA, March 1983.

5. Morawski, Paul E., Richard O. Nugent, and Richard W. Tucker, *TAC-1: A Knowledge-Based Air Force Tactical Battle Management Testbed*, MTR-87W00171, The MITRE Corporation, McLean VA, September 1987.